Make Linux/Unix Script Portable With #!/usr/bin/env As a Shebang

Author: Vivek Gite • Last updated: August 12, 2020 • 28 comments

You may notice that most shell and Perl script starts with the following line:

```
#!/bin/bash
```

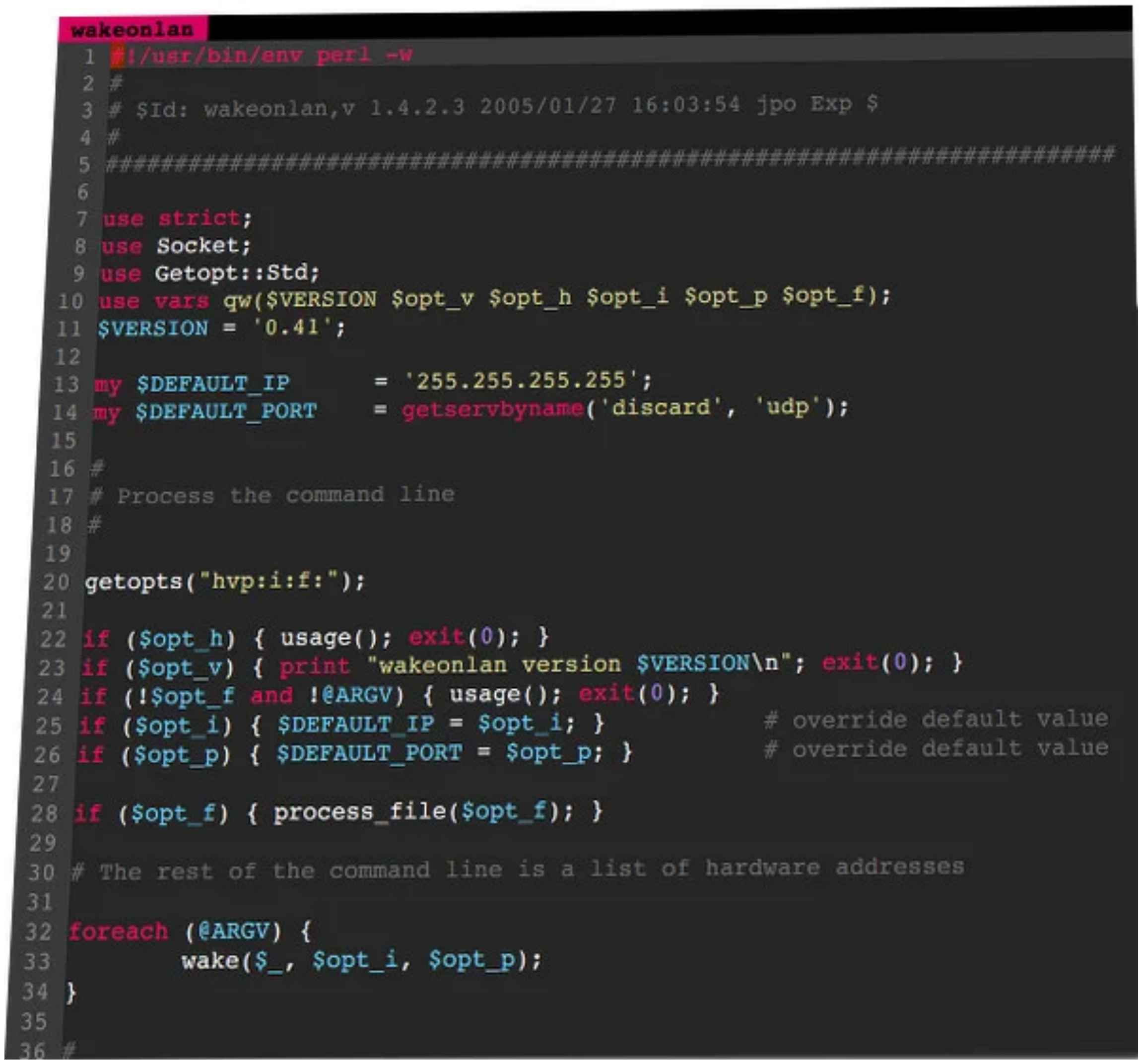
OR

```
#!/usr/bin/perl
```

Let us find out why is it a good idea to use `#!/usr/bin/env bash` instead of `#!/bin/bash` as shebang?

#!/usr/bin/env As a Shebang

The `#!` is called a shebang. It consists of a number sign and an exclamation point character (#!), followed by the full path to the interpreter such as /bin/bash. All scripts under Linux, *BSD, macOS, and Unix-like system execute using the interpreter specified on a first line. However, there is a small problem. BASH or Perl is not always in the same location (PATH) such as /bin/bash or /usr/bin/perl. If you want to make sure that script is **portable** across different UNIX like operating systems you need to use /usr/bin/env command as shebang.



```
wakeonlan
#!/usr/bin/env perl -w
#
# $Id: wakeonlan,v 1.4.2.3 2005/01/27 16:03:54 jpo Exp $
#
########################################################################

use strict;
use Socket;
use Getopt::Std;
use vars qw($VERSION $opt_v $opt_h $opt_i $opt_p $opt_f);
$VERSION = '0.41';

my $DEFAULT_IP      = '255.255.255.255';
my $DEFAULT_PORT    = getservbyname('discard', 'udp');

#
# Process the command line
#

getopts("hvp:i:f:");

if ($opt_h) { usage(); exit(0); }
if ($opt_v) { print "wakeonlan version $VERSION\n"; exit(0); }
if (!$opt_f and !@ARGV) { usage(); exit(0); }
if ($opt_i) { $DEFAULT_IP = $opt_i; }           # override default value
if ($opt_p) { $DEFAULT_PORT = $opt_p; }         # override default value

if ($opt_f) { process_file($opt_f); }

# The rest of the command line is a list of hardware addresses

foreach (@ARGV) {
        wake($_, $opt_i, $opt_p);
}

#
```

Fig.01: My Perl script start with /usr/bin/env

Make Linux/Unix Script Portable With #!/usr/bin/env As a Shebang

The env command allows to run a program in a modified environment. First, find line

```
#!/bin/bash
```

Replace with

```
#!/usr/bin/env bash
```

For example here is a small bash shell script:

```
#!/usr/bin/env bash
x=5
y=10
echo "$x and $y"
```

The advantage of `#!/usr/bin/env bash` is that it will use whatever bash executable appears first in the running user's $PATH variable. If you have two version of bash installed as follows and PATH set to /home/vivek/bin:/usr/local /bin:/usr/bin:/bin:/usr/games/bin:/bin:/usr/bin, than bash4 will execute the script:

- `/bin/bash` # <-- bash3
- `/usr/local/bin/bash` # <-- bash4

Using env in the shebang of a script

env is a shell command for Unix-like operating systems. We can use it for various purposes. For example:

- Display a list of environment variables

- Run another command in a modified environment
- Add or remove variables
- Change the value of existing variables

To show current environment variables defined by your shell, run env command:

```
env
```

We can filter out infromation using the grep command/egrep command as follows:

```
env | grep HOME
env | egrep 'HOME|USER|VERSION|SHELL|PWD'
```

```
SHELL=/bin/bash
GNOME_SHELL_SESSION_MODE=ubuntu
PWD=/tmp
HOME=/home/vivek
USERNAME=vivek
VTE_VERSION=6003
USER=vivek
OLDPWD=/tmp
```

Let us see some more examples.

Perl example

```
#!/usr/bin/env perl
use warnings;
```

```
print "Hello " x 5;
print "\\n";
```

Python2.x example:

```
#!/usr/bin/env python
x=10
y=20
z=x+y
print z
```

Another example with python3:

```
#!/usr/bin/env python3
import boto3
t = boto3.resource('s3')
for b in t.buckets.all():
     print(b.name)
```

Is env always located at /usr/bin/env

You can use the type command or command command to locate exact path for the env command:

```
type env
command -V env
```

Sample outputs:

```
env is /usr/bin/env
```

Some info from various *nix systems:

OS	env path
OpenBSD	/usr/bin/env
FreeBSD	/usr/bin/env
Debian	/usr/bin/env
Ubuntu	/usr/bin/env
CentOS	/usr/bin/env
macOS	/usr/bin/env
SUSE	/usr/bin/env
RHEL	/usr/bin/env
NetBSD	/usr/bin/env
Solaris	/usr/bin/env

Use shellcheck

ShellCheck is a static analysis tool for shell scripts. One can use it to finds bugs in your shell scripts. It is written in Haskell. You can find warnings and suggestions for bash/sh shell scripts with this tool. Let us see how to install and use ShellCheck on a Linux or Unix-like system to enhance your shell scripts, avoid errors and productivity.

Conclusion

I just made my Linux/Unix script portable with `#!/usr/bin/env`. The env-mechanism is hugely enhancing convenience, and almost all Unix like systems today provide /usr/bin/env. I regular use FreeBSD, CentOS/RHEL/Ubuntu /Debian/SUSE Linux, macOS, and OpenBSD and all of them provided /usr/bin/env by default. Now you do not have to search for a program via the PATH environment variable. It makes the script more portable. Also, note that it is not a foolproof method. Always make sure you have /usr/bin/env exists or use a soft link/symbolic link to point it to correct path for env command. Moreover, your work or script looks more professional with this simple hack. See env command man page (documentation) or type the following man command:

```
$ man bash
$ man env
```

Related Posts

Apache give each user their own cgi-bin directory

FreeBSD portupgrade…

How To Build Secure and Portable Linux Based System

Book review: Secrets of the PlayStation Portable

Asus Ebox Ultra portable Desktop PC

Download FireFox 3 Portable USB Pen / Device Friendly…

How Do I Make Linux / UNIX Filesystem Backup With dd?

Linux / UNIX Software that make retrieving the data more…

Category	List of Unix and Linux commands
Ansible	Check version • Fedora • FreeBSD • Linux • Ubuntu 18.04 • Ubuntu • macOS
Download managers	wget
Driver Management	Linux Nvidia driver • lsmod
Documentation	help • mandb • man • pinfo
Disk Management	df • duf • ncdu • pydf

Category	List of Unix and Linux commands
File Management	cat • cp • less • mkdir • more • tree
Firewall	Alpine Awall • CentOS 8 • OpenSUSE • RHEL 8 • Ubuntu 16.04 • Ubuntu 18.04 • Ubuntu 20.04
KVM Virtualization	CentOS/RHEL 7 • CentOS/RHEL 8 • Debian 9/10/11 • Ubuntu 20.04
Linux Desktop apps	Chrome • Chromium • GIMP • Skype • Spotify • VLC 3
Modern utilities	bat • exa
Network Utilities	NetHogs • dig • host • ip • nmap • ping
OpenVPN	CentOS 7 • CentOS 8 • Debian 10 • Debian 11 • Debian 8/9 • Ubuntu 18.04 • Ubuntu 20.04
Power Management	upower
Package Manager	apk • apt-get • apt • yum
Processes Management	bg • chroot • cron • disown • fg • glances • gtop • iotop • jobs • killall • kill • pidof • pstree • pwdx • time • vtop
Searching	ag • egrep • grep • whereis • which
Shell builtins	compgen • echo • printf
System Management	reboot • shutdown
Terminal/ssh	tty
Text processing	cut • rev
User Environment	exit • who
User Information	groups • id • lastcomm • last • lid/libuser-lid • logname • members • users • whoami • w
User Management	/etc/group • /etc/passwd • /etc/shadow • chsh

Category	List of Unix and Linux commands
WireGuard VPN	Alpine • Amazon Linux • CentOS 8 • Debian 10 • Firewall • Ubuntu 20.04 • qrencode

28 comments… add one ↓

bhaskar • Mar 7, 2007 @ 15:15

I am confused.
Why is /usr/bin/env more portable than /bin/bash.
Besides on most linux distros bash is found in /bin, but env is not guaranteed to be found under /usr/bin.
e.g. in my case (FC5) env is under /bin.

My point is , which ever way you need to know the absolute path to either env, or bash, so why bother ?

reply link

nixCraft • Mar 7, 2007 @ 16:27

If you move from Linux distro to BSD you will see bash is located at /usr/local /bin/bash OR to Solaris you will see bash at /opt or some other location. Instead of adjusting all the location admin can create a /usr/bin/env softlink and problem solved. Just imagine you have 100s of shell and perl scripts…

This is not just about Linux. It is about running a script under different UNIX like oses.

reply link

nag • Jul 12, 2007 @ 21:43

Thanks for “shebang” and its explanation. Its really helpful.

reply link

IHar Filipau • Aug 2, 2007 @ 8:07

The only problem I have with env, that on some systems lacking ‘use warnings’, I can’t pass ‘-w’ on command line. Or is it possible somehow?

reply link

nixCraft • Sep 6, 2007 @ 9:44

You can pass /bin/path/to/mybinary -w

reply link

Carsten • Jan 14, 2009 @ 8:45

I do not feel very comfortable with the security aspects of this hack.
It may be more portable but it reduces control.
Bash and other shells are hardened so that they can be used for admin jobs withstanding tampering efforts by non root users.
Using the described “env” solution may introduce vulnerabilities which are difficult to oversee or analyze.

Instead of linking env and using it to deal with compatibity I rather add compatibility links to my systems which link bash or perl to a uniform location e.g. ln -s /usr/local/bin/bash /bin/bash

reply link

Terrible • Jan 26, 2009 @ 22:00

Terrible advice. Do not do this. Portability should stem from your installion routines, not some security and design problem causing hack.

reply link

robsku • Dec 17, 2010 @ 23:29

@Terrible

You could be correct, but I'd like to hear your opinion of why exactly is it more harmful?

Well, it runs the command, perl for example, like you would run perl by hand – so I assume it searches via PATH – I can see that individual user may then create executable 'perl' of his own under his home directory and change PATH so that env will call that… However that already means that user in question is doing this on purpose, in which case he could just call the harmful program himself – no need to even include the script, perl nor env on that and if done that way it would not affect other users which would still get to run the script as intended.

However security and holes in it can be complex issues and I'm not a professional at all… If indeed using env is more of a possible security issue than creating shebang pointing right to correct interpreter in install routine I would love to learn why exactly is it safer, about possible security issues in using env, etc.

I would think that when installed system wide both ways would be as safe – and one trying to do harm could install the whole software locally under his home directory anyway but could not alter what is ran when other users call the script in question – not understanding makes me even more curious, but most importantly I want to ensure security…

reply link

VirtualSMF • Aug 26, 2016 @ 13:51

Imagine a trojan ‘bash’ placed in ‘/tmp’.. An sa with sudo all nopasswd.. Their path has ‘.’ at the beginning.. CWD as ‘/tmp’.. Now imagine them executing any of their scripts with this hack?

That’s a lot of stars to align, but remember.. you only need this to happen once! The patient will be rewarded!

reply link

palunon • Sep 22, 2016 @ 12:06

>Imagine a trojan ‘bash’ placed in ‘/tmp’.. An sa with sudo all nopasswd.. Their path has ‘.’ at the beginning.. CWD as ‘/tmp’.. Now imagine them executing any of their scripts with this hack?

How did a trojan arrive in /tmp ?

Their path has ‘.’ at the beginning.. -> Call you trojan ls and you have way more chance for this to work… Don’t put ‘.’ in PATH.

reply link

VirtualSMF • Sep 27, 2016 @ 12:48

I couldn’t agree more.. again, just giving an example.. no matter how implausible.

reply link

someone • Aug 13, 2011 @ 19:01

Even the official python tutorial website is using:
#! /usr/bin/env python
http://docs.python.org/tutorial/interpreter.html

reply link

Triston J. Taylor • Dec 24, 2012 @ 18:11

Here's an idea, instead of wasting cpu and peoples time, suggest a link to /bin/bash always. This stupid idea of using env command is a waste of the unix operating system.

Symbolic links were created to solve problems exactly like this. Some idiot came up with the great idea to make it more complicated by suggesting a link to bin env… I swear this is absolutely retarded.

Quit being lazy and put a proper link in /bin. What is the point in spawning processes you don't need? More professional? According to who? The idiots who never wrote a single program of assembler code in their life? The idiots who couldn't write a bash script to remove duplicate words from a list without resorting to external commands?

Oh yeah, you must mean those idiots…. The one's who couldn't code their way out of wet paper sack.

reply link

Triston J. Taylor • Dec 24, 2012 @ 18:16

And another thing, if its really that important to be portable, and you can't create a link in /bin then your scripts need to be auto-generated on-site with the proper header line.

Quit wasting resources and practice efficiency. Wastefulness is a bad habit that not only adds up, but is certain to be a perpetual mistake.

reply link

Anon • Mar 21, 2014 @ 0:24

Are you seriously saying that a person is an idiot just because of this? You look like one acting like you are the only one who knows… Just saying.

reply link

Ian Epp • Jul 17, 2013 @ 21:16

@Triston J. Taylor

I came here to solve a problem with the python interpreter changing based on which python virtual environment was activated at a given time. Remapping the symlink doesn't work at all as I may have one shell in one environment and another in a completely different one – it invokes different python symlinks. Using env fixes this perfectly.

reply link

Rodney • Sep 20, 2013 @ 19:50

One problem with /usr/bin/env SomeCmd is you can't pass additional arguments.

So you can't /usr/bin/env SomeCmd -someArg -someOtherArg

Why? Because /usr/bin/env doesn't support passing those as arguments on same platforms.

One hack to work around this is

reply link

Rodney • Sep 20, 2013 @ 19:52

Also see

This explains that the shebang only accepts 1 argument

reply link

Ovidiu • Jan 31, 2014 @ 16:25

It is not a portable solution. On Windows you could have C:cygwin64binenv.exe instead of /usr/bin/env.

So assuming the path of env is not a portable solution.

reply link

snke • Apr 22, 2014 @ 5:57

I just want to add to this now seven-year-old conversation that modern developers often change their development environment depending on which project they are working on.

For example using rvm you might often change which version of the ruby interpreter you are using to ensure compatibility across different versions or to work on legacy code. Switching rubies switches your PATH to point to whichever ruby you are using.

Additionally a developer may want to develop in a different version of say, python, than the version that ships with OS X, but doesn't want to overwrite the system's version or uninstall it because they know they do not know what the consequences of doing so. Using a package manager like Homebrew makes it easy to install scripting languages for the local user rather than forcing you to install it as root. Is that not more secure?

In each of these scenarios you want to be using /usr/bin/env in your scripts because hard-coding the binary location is not practical and will be irritating to anyone else on your team if you try and pass off a script with something like #!/Users/joejimbob/.rvm/rubies/ruby-1.9.3-p286/bin/ruby at the top. Saying MY script doesn't work because YOU don't know how to symlink might earn me 3 or 4 old school points, but I would still be a jerk.

As for being wasteful, trading end-user CPU cycles for a shorter development time and broader compatibility is the world we live in. Rejecting this and calling people lazy for it, is an obsolete way of thinking and shows a distrust of modern methods and software which is the sign of a bad coder (see #6 here: http://www.yacoset.com/Home/signs-that-you-re-a-bad-programmer). Isn't abstraction the way by which we accomplish the most amazing things with technology? Isn't it more wasteful to waste development time by re-inventing the wheel and forcing your peers to understand YOUR style rather than what's most practical?

I inherited a system from such old-fashioned types of people and I can tell you I still haven't unearthed all of their "gotchas" of absolute paths and i-dont-trust-this-new-language-so-im-going-to-be-clever types of garbage. The way they did things was to approach the problem as if they were the first person to ever tackle it or that at the very least, they knew better because they've been around longer. I swear these guys thought they were going to be buried with the server.

reply link

Pete • Apr 23, 2014 @ 14:56

@Ovidiu

Well you have to start somewhere! For the most part *nix distributions have standardized the locations so that both /usr/bin/env and /bin/env exist.

@snke
Totally agree. I came here because I am faced with Perl code that starts "#!

/usr/bin/perl”, but I have a requirement that means using a later release of Perl than that installed in the system area and in the environment I work in we are not allowed to change the “system” perl or add to its modules. So it should have been coded “#! /usr/bin/env perl”.

Thanks for the page you linked to.

reply link

Matt D. • Oct 14, 2014 @ 11:29

I would also like to add that this technique has allowed my scripts to be portable to OSX where bash is still stuck at version 3. For example, I’ve installed mac ports and set my path for ‘bash’ to take precedence. But if I were to hard-code #!/bin/bash then that would do no good.

@Pete

‘which env’ on OSX 10.9 /usr/bin/env as the only result and appears to be the most portable between Cygwin, Linux, and OSX.

reply link

Piotr Dobrogost • Jan 7, 2015 @ 15:10

Funny thing is that according to `man env` “env – run a program in a modified environment”. So running `#!/usr/bin/env bash` instead of `#!/bin/bash` should have no effect as there are no environment variables being set or unset. I guess when `env` runs a command passed to it the PATH environment variable is used to locate the command but why there’s no information about this in the man page of `env` is a mystery for me.

reply link

Anon • Aug 26, 2016 @ 20:53

The example on the image generally doesn't work (#!/usrbin/env perl -w).
See: http://unix.stackexchange.com/questions/63979/shebang-line-with-usr-bin-env-command-argument-fails-on-linux

reply link

Dave • Feb 27, 2017 @ 12:12

https://github.com/freerobby/wakemachine/blob/master/wakeonlan.pl

reply link

Wellington Torrejais da Silva • Jun 12, 2017 @ 12:48

Nice tip!

reply link

---

akizminet • Aug 11, 2020 @ 6:34

There's probably a mistake in your command to show current environment variables. \`evn\` instead of \`env\`

reply link

> **Vivek Gite** • Aug 12, 2020 @ 19:16
>
> Thanks for the heads up!
>
> reply link

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name

Email

Website

Post Comment

Use HTML <pre>...</pre> for code samples. Your comment will appear only after approval by the site admin.

Next post: March 6, 2007 : nixCraft FAQ Roundup

Previous post: OpenSolaris shipping free DVDs all around the world

SEARCH

To search, type & hit enter...

1 30 Cool Open Source Software I Discovered in 2013

2 30 Handy Bash Shell Aliases For Linux / Unix / Mac OS X

3 Top 32 Nmap Command Examples For Linux Sys/Network Admins

4 25 PHP Security Best Practices For Linux Sys Admins

5 30 Linux System Monitoring Tools Every SysAdmin Should Know

6 40 Linux Server Hardening Security Tips

7 Linux: 25 Iptables Netfilter Firewall Examples For New SysAdmins

8 Top 20 OpenSSH Server Best Security Practices

9 Top 25 Nginx Web Server Best Security Practices

10 My 10 UNIX Command Line Mistakes

SIGN UP FOR MY NEWSLETTER

- → Howtos & Tutorials
- → Linux shell scripting tutorial
- → RSS/Feed
- → About nixCraft